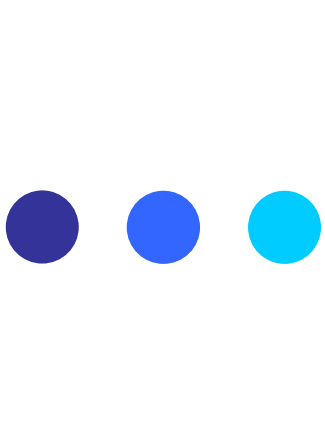


PB153 Operační systémy a jejich rozhraní

Principy výstavby OS

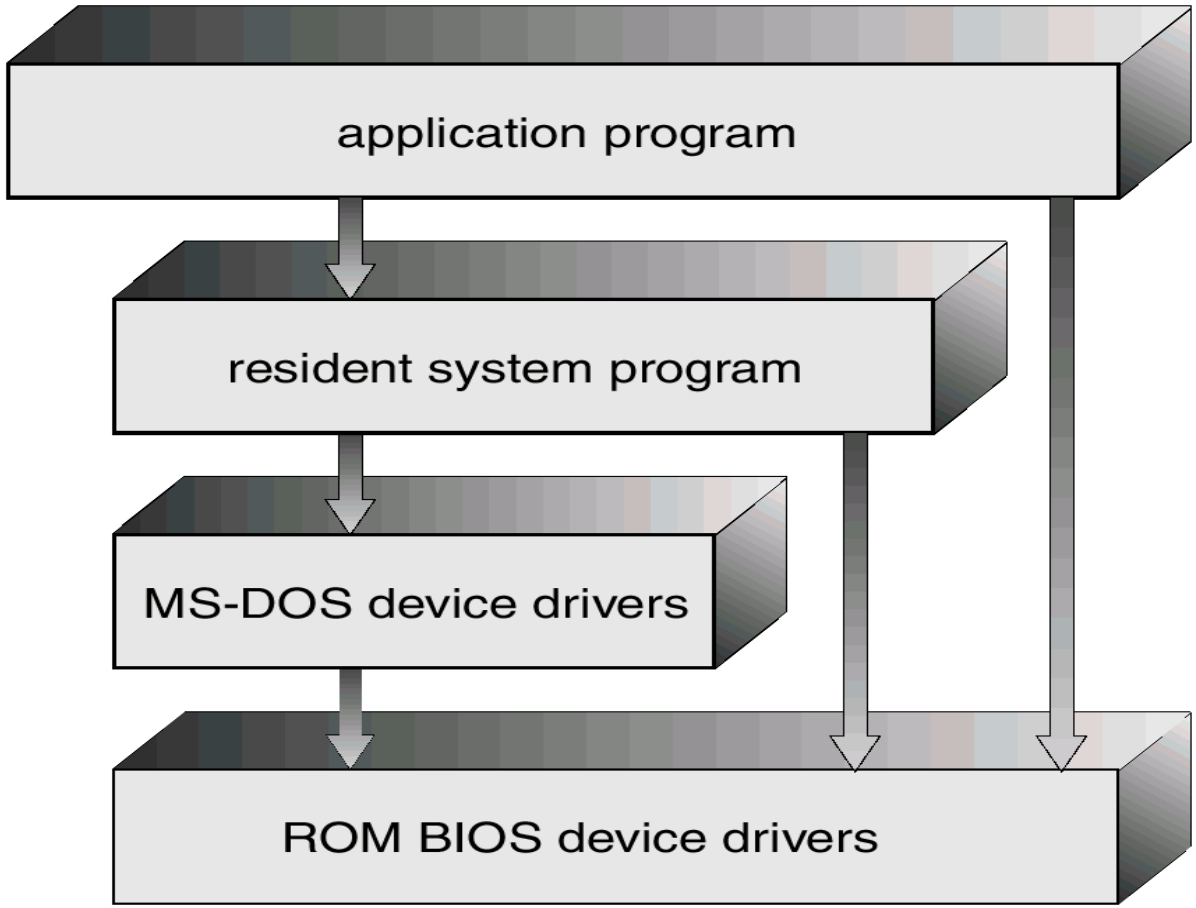
Systémové programy

- Poskytují pohodlné prostředí pro vývoj a spouštění programů.
- Typická kategorizace systémových programů
  - Práce se soubory, kopírování, editace, …
  - Získávání, definování a údržba systémových informací
  - Modifikace souborů
  - Podpora jazykových prostředí
  - Zavádění a provádění programů
  - Komunikace
- Významným systémovým programem je shell
  - má shell implementovat příkazy sám nebo jen volat externí programy?
- Uživatelský pohled na OS je dán především systémovými programy a ne voláními systému (system calls)

Vnitřní struktura OS

- Existuje řada přístupů a implementací
  - jedno velké monolitické jádro
  - modulární, hierarchický přístup
  - malé jádro a samostatné procesy
- Struktura mnoha OS je poznamenána historií OS a původními záměry, které se mohou od současného stavu radikálně lišit

Příklad: MS-DOS



Příklad MS-DOS (2)

- Při programování pro OS MS-DOS využíváme služeb
  - spuštěných rezidentních programů
    - např. ovladač myši (poskytuje služby na INT 33h)
  - operačního systému
    - např. přístup k souborům (INT 21h)
  - BIOSu
    - např. nastavení grafického režimu (INT 10h)
  - přímo HW
    - např. přímo zápis do videopaměti pro zobrazení dat

Příklad MS-DOS (3)

- Změna fontů v textovém režimu (bez využití služeb BIOSu, OS, přímo HW)

```
asm cli;
outport(0x3c4,0x0402);
outport(0x3c4,0x0704);
outport(0x3ce,0x0204);
outport(0x3ce,0x0005);
outport(0x3ce,0x0406);
for(i=0;i<=254;i++)
{ for(j=0;j<=15;j++)
{p=MK_FP(0xa000,32*i+j);
*p=font[y]; y++; } }
outport(0x3c4,0x0302);
outport(0x3c4,0x0304);
outport(0x3ce,0x0004);
outport(0x3ce,0x1005);
outport(0x3ce,0x0e06);
asm sti;
```

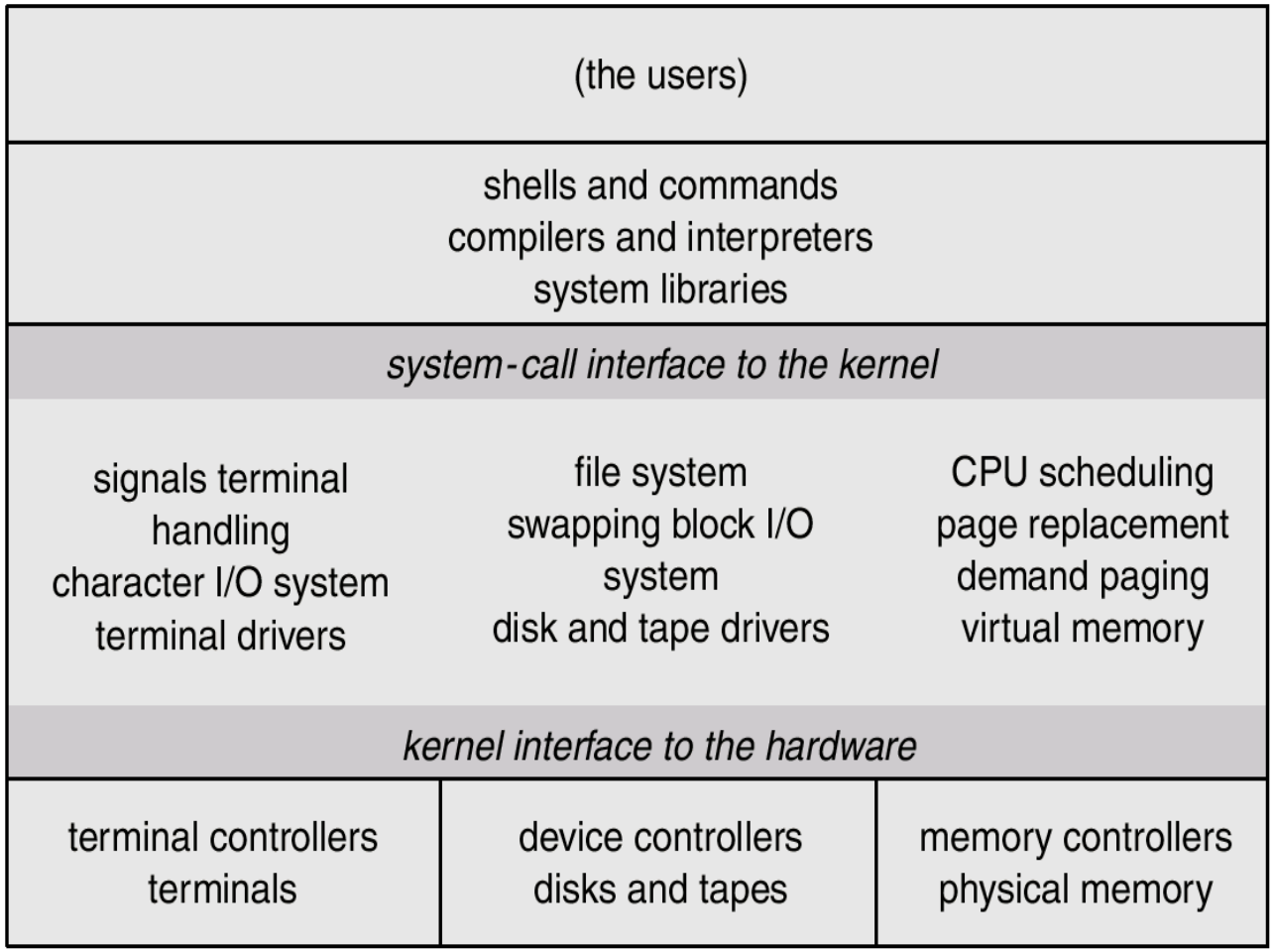
Příklad MS-DOS (4)

- Hlavní cíl návrhu
  - maximální možná funkcionalita v co nejmenším prostoru
- Výsledek
  - modulová architektura není aplikovaná
  - i když MS-DOS má jistou strukturu, jeho rozhraní a jednotlivé komponenty nejsou důsledně separovány a uspořádány

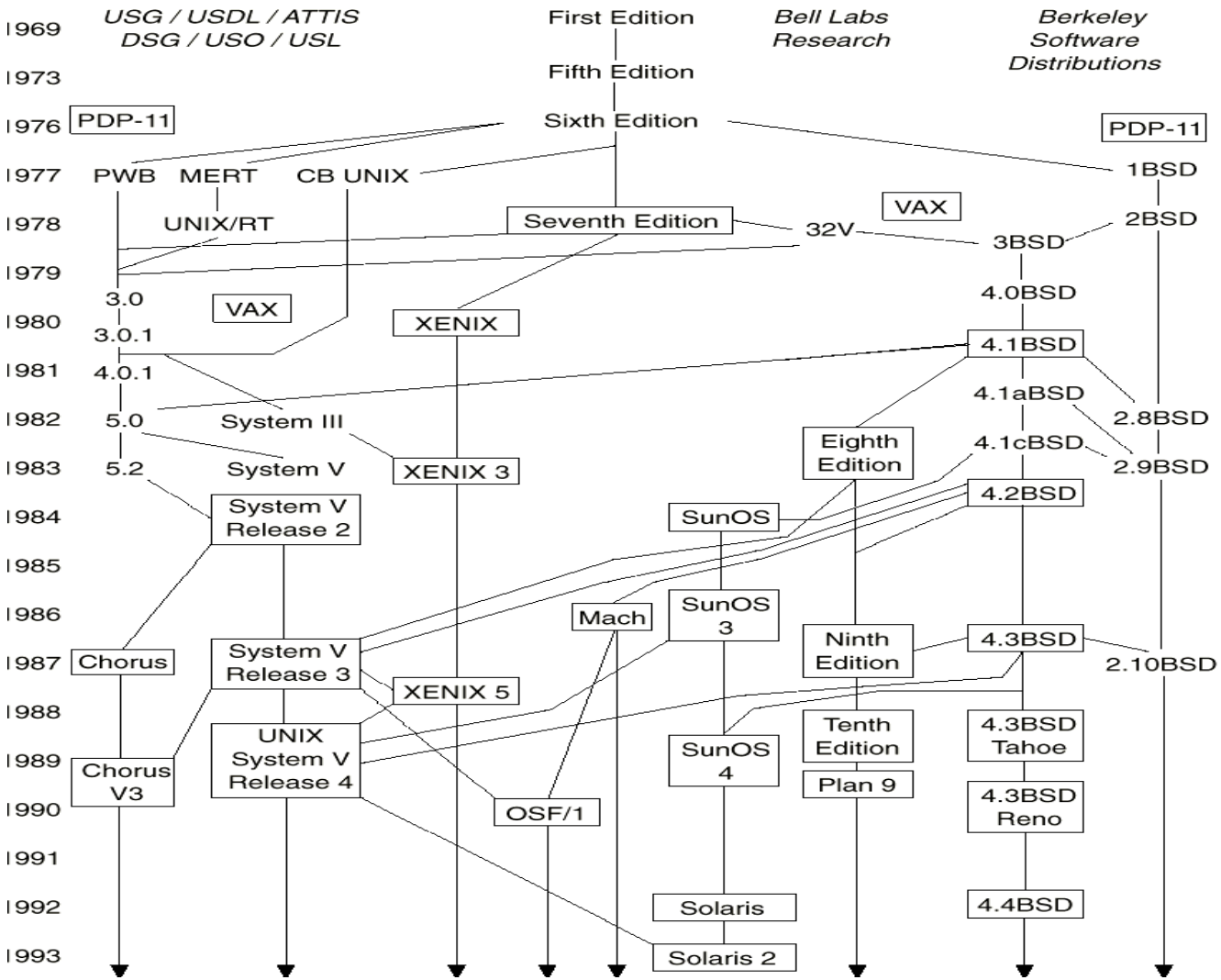
Příklad UNIX

- Také omezen hardwarem
  - vznik v polovině 70. let
- OS Unix sestává ze 2 částí
  - systémové programy
  - jádro
    - vše, co se nachází pod rozhraním volání systému a nad fyzickým hardware
    - obstarává plnění funkcí z oblastí systému souborů, plánování CPU, správy paměti, …
    - vrstvová architektura sice existuje, ale hodně funkcí je na jedné úrovni

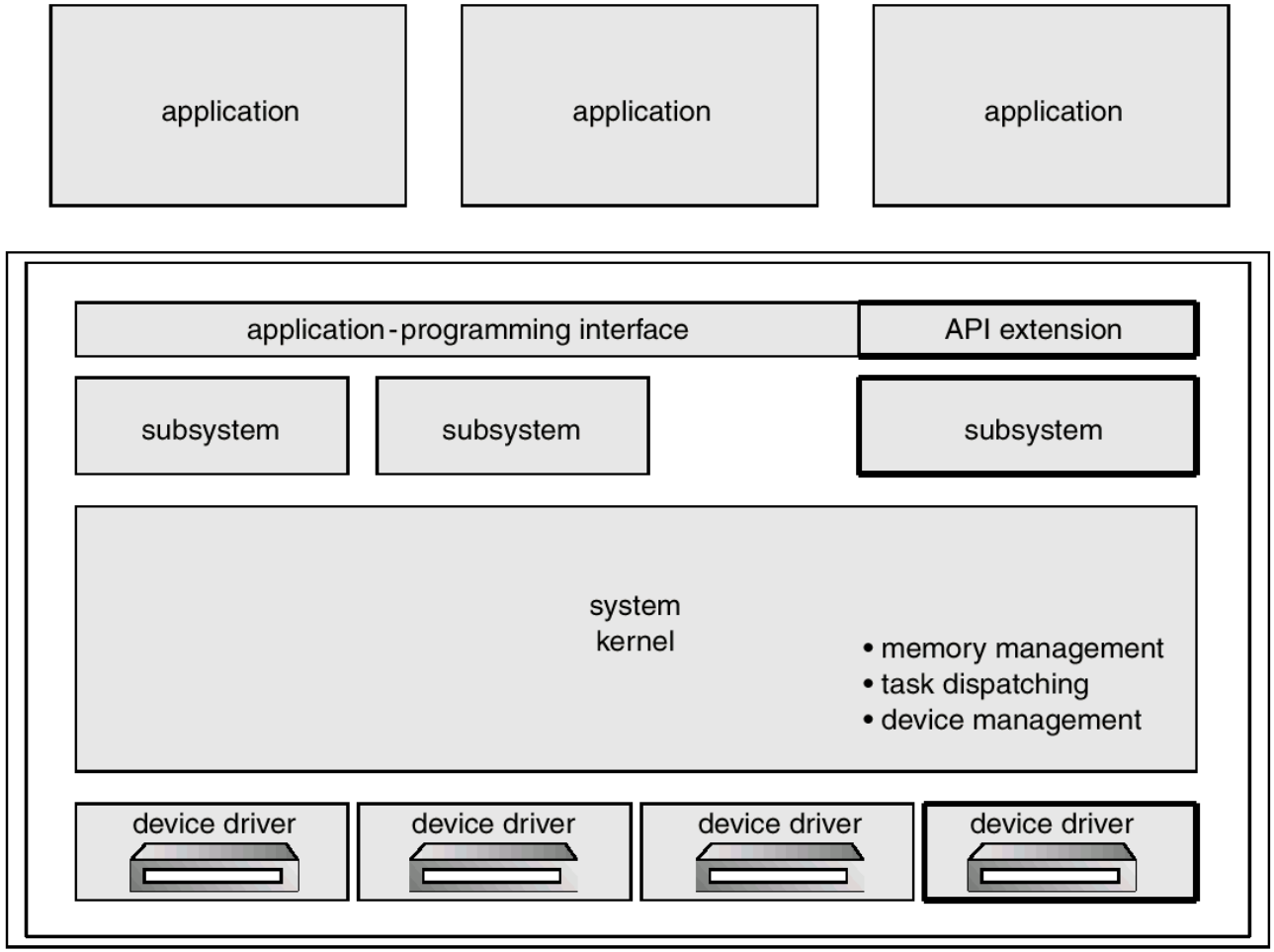
Příklad UNIX (2)



Příklad: UNIX (3)



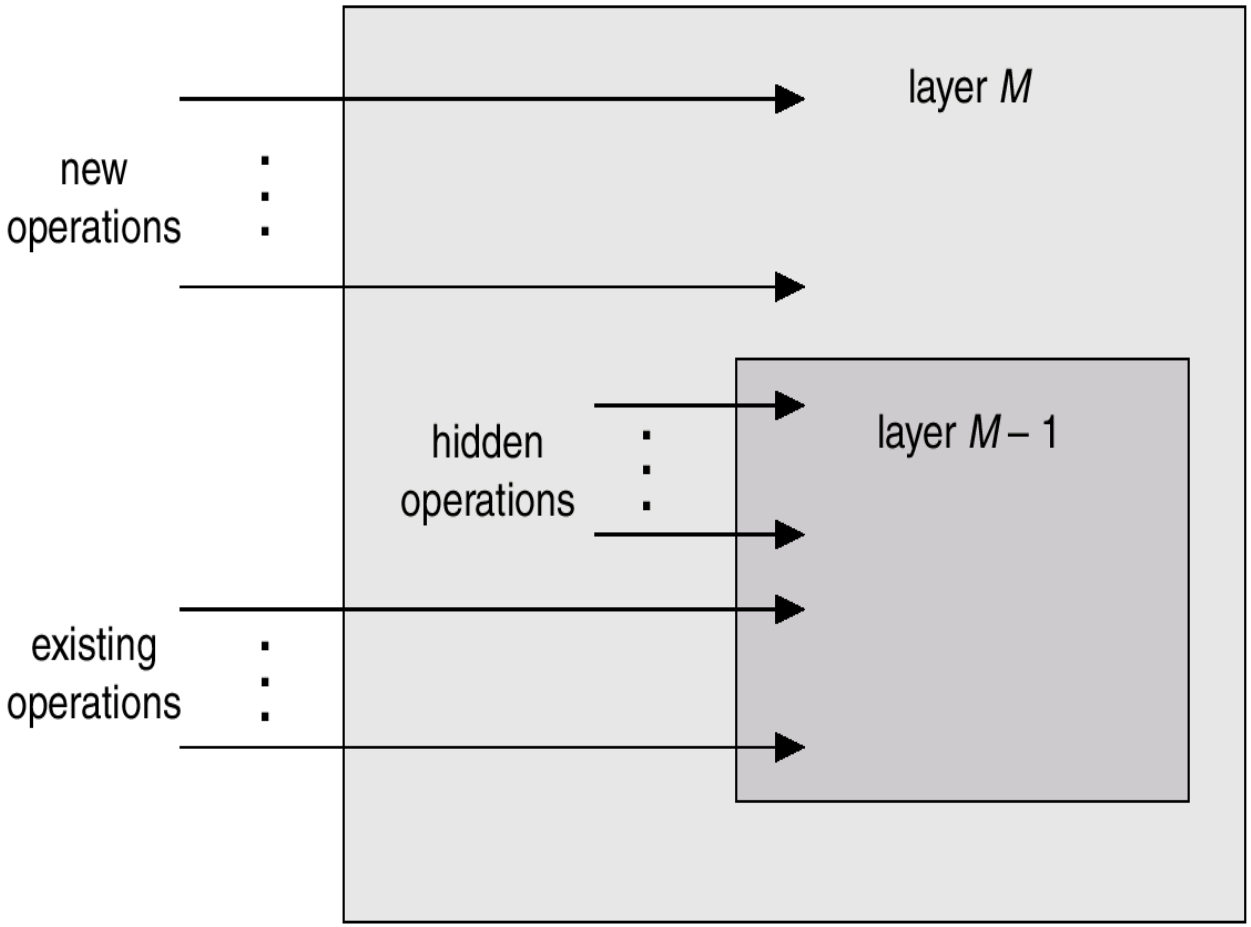
Příklad OS/2


application
application
application
application - programming interface
API extension
subsystem
subsystem
subsystem
system
kernel
• memory management
• task dispatching
• device management
device driver
device driver
device driver
device driver

Hierarchická vrstvová architektura

- OS se dělí do jistého počtu vrstev (úrovní)
- Každá vrstva je budována na funkcionalitě nižších vrstev
- Nejnižší vrstva (0) je hardware
- Nejvyšší vrstva je uživatelské rozhraní
- Pomocí principu modulů jsou vrstvy vybírány tak, aby každá používala funkcí (operací) a služeb pouze vrstvy $n - 1$

Hierarchická architektura



Hierarchická struktura

- Řeší problém přílišné složitosti velkého systému
  - Provádí se dekompozice velkého problému na několik menších zvládnutelných problémů
- Každá úroveň řeší konzistentní podmnožinu funkcí
- Nižší vrstva nabízí vyšší vrstvě „primitivní“ funkce (služby)
- Nižší vrstva nemůže požadovat provedení služeb vyšší vrstvy
- Používají se přesně definovaná rozhraní
  - Jednu vrstvu lze uvnitř modifikovat, aniž to ovlivní ostatní vrstvy – princip modularity

Hierarchická struktura

- Výhodou je modularita OS
- Nevýhodou je především vyšší režie a tím pomalejší vykonávání systémových volání
- Protože efektivita hraje v jádře OS významnou roli je třeba volit kompromis
  - pouze omezený počet úrovní pokrývající vyšší funkcionalitu
  - příklad: první verze Windows NT měli hierarchickou strukturu s řadou vrstev, avšak pro zvýšení výkonu OS bylo ve verzi NT 4.0 rozhodnuto přesunout více funkcionality do jádra a sloučit některé vrstvy

Provádění služeb v klasickém OS

- Klasický OS (non-process kernel OS)
  - OS je prováděn jako samostatná entita v privilegovaném režimu
  - procesy – jen uživatelské programy
- Služba se provádí jako součást jádra
- Služba se provádí v rámci procesů
  - obecně lze celý OS provádět v kontextu uživatelského procesu
    - Leží v jeho adresovém prostoru
  - přerušení (volání služby OS) vyvolává implicitně pouze přepnutí režimu procesoru (z uživatelského do privilegovaného), ne změnu kontextu
  - k přepínání kontextu procesů dochází jen tehdy, je-li to nutné z hlediska plánování
  - pro volání procedur v rámci jádra se používá samostatný zásobník
  - program a data OS jsou ve sdíleném adresovém prostoru a sdílí je všechny uživatelské procesy

Služby v procesově konstruovaném OS

- OS je souhrnem systémových procesů
- Jádro tyto systémové procesy separuje, ale umožňuje jim synchronizaci a komunikaci
- Snaha o provádění co nejmenší části kódu v privilegovaném režimu procesoru
- V krajním případě je jádro pouze ústředna pro přepojování zpráv
- Takové řešení OS je snadno implementovatelné na multiprocesorových systémech
- Malé jádro - mikrojádro

Struktura s mikrojádrem

- Microkernel System Structure
- Malé jádro OS plnící pouze několik málo nezbytných funkcí
  - primitivní správa paměti (adresový prostor)
  - komunikace mezi procesy – Interprocess communication (IPC)
- Většina funkcí z jádra se přesouvá do „uživatelské“ oblasti
  - ovladače HW zařízení, služby systému souborů, virtualizace paměti …
  - mezi uživatelskými procesy se komunikuje předáváním zpráv

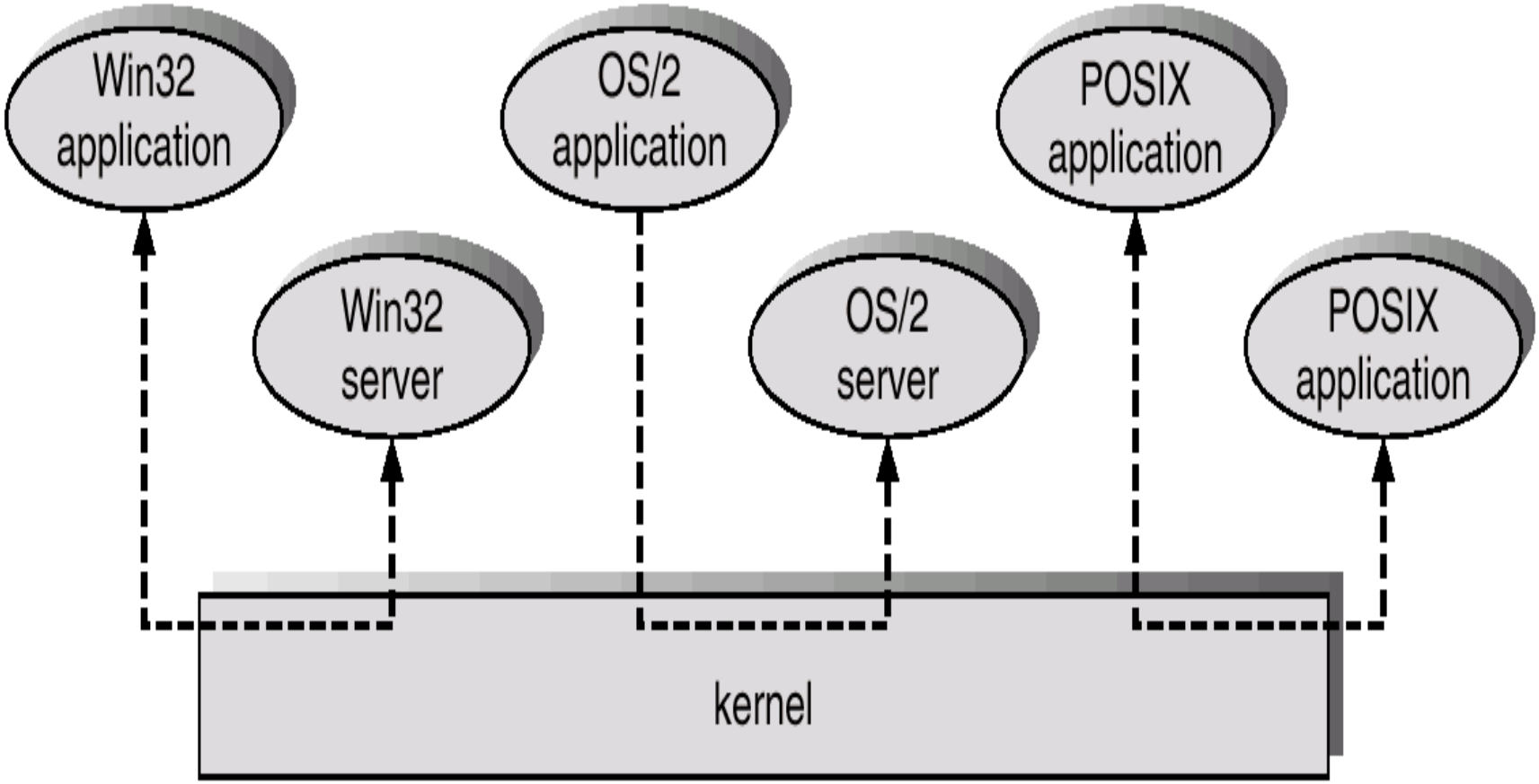
Struktura s mikrojádrem (2)

- Výhody mikrojádra
  - snadná přenositelnost OS, jádro je malé
  - vyšší spolehlivost (moduly mají jasné API a jsou snadněji testovatelné)
  - vyšší bezpečnost (méně kódu OS běží v režimu jádra)
  - flexibilita (jednodušší modifikace, přidání, odebrání modulů)
  - všechny služby jsou poskytovány jednotně (výměnou zpráv)
- Nevýhoda mikrojádra
  - zvýšená režie
    - volání služeb je nahrazeno výměnou zpráv mezi procesy

Mach

- Klasickým příkladem OS s mikrojádrem je Mach vytvořený v 80. letech
- Na přístupu Mach je založen např. Tru64 UNIX nebo realtimový OS QNX
- Windows NT používají hybridní strukturu
  - jádro má vrstevnou strukturu a zajišťuje komunikaci aplikace se „servery“
  - pro jednotlivé typy aplikací (Win32, OS/2, POSIX) existují „servery“ běžící v uživatelském režimu

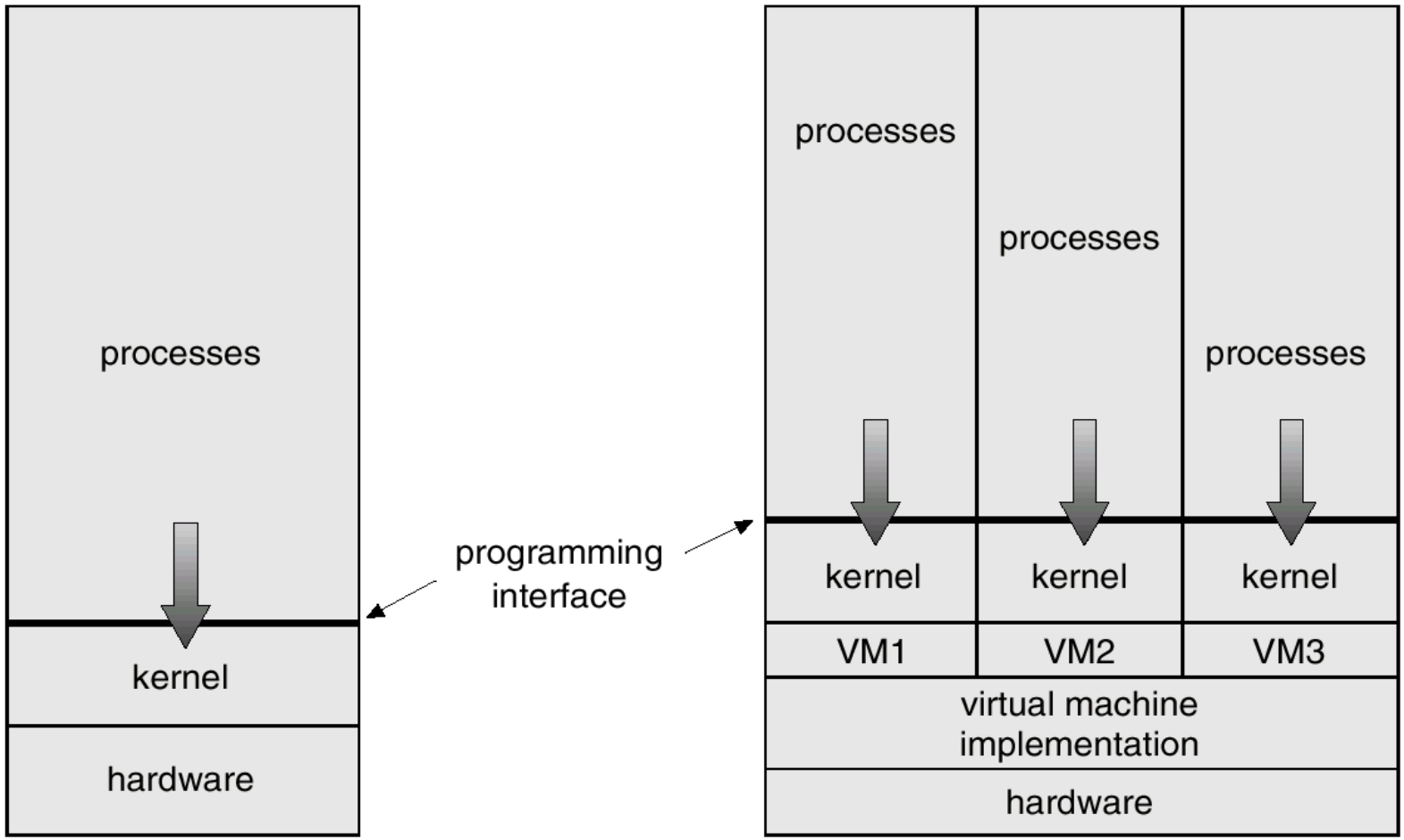
Příklad: Windows NT



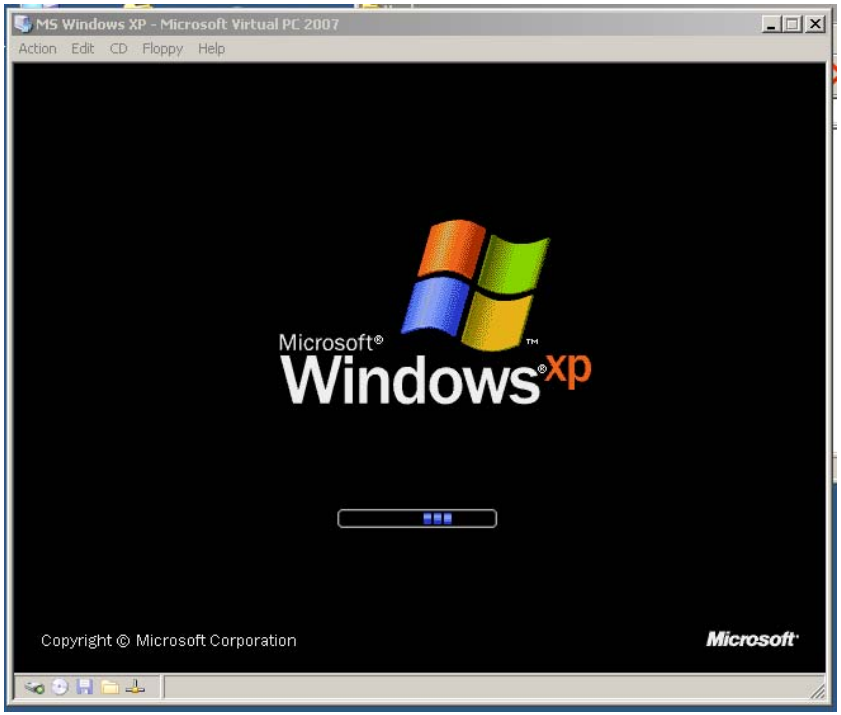
Virtuální stroje

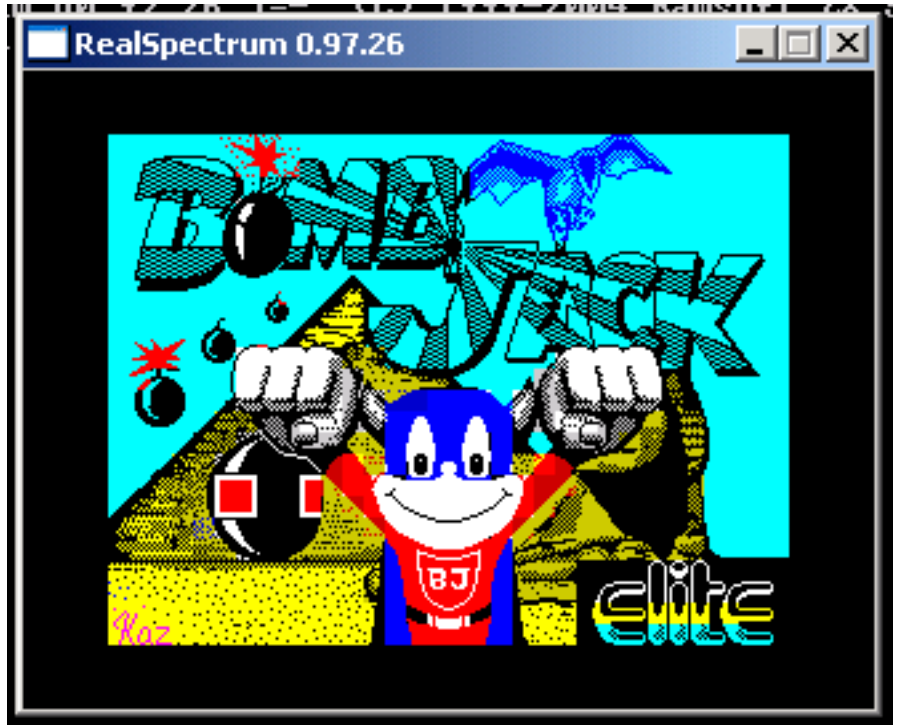
- Virtuální stroj vytváří rozhraní identické s původním holým hardwarem
- Operační systém vytváří iluzi více procesů, kde každý běží na svém vlastním procesoru se svou vlastní (virtuální) pamětí
  - může vytvářet iluzi více systémů
    - s vlastnostmi původního fyzického počítače nebo úplněji jiného stroje
  - každý uživatel na jediném stroji může tudíž provozovat jiný OS
  - virtuální stroje zajišťují úplnou ochranu systémových zdrojů
    - každý virtuální stroj je izolován od všech ostatních virtuálních strojů
    - to však ztěžuje komunikaci mezi virtuálními stroji
  - Ve virtuálním stroji může běžet virtuální stroj ☺
    - Usnadňuje to např. ladění OS během jeho provozování
- Implementace virtuálních strojů není triviální

Virtuální stroje (2)



Virtualizace (příklad)





Java Virtual Machine

- Prostředí Java dosahuje nezávislosti na použité HW platformě pomocí JVM (Java Virtual Machine)
  - pro každou platformu je implementován JVM
  - kompilátor Java (javac) generuje „bytecode“ (ne přímo strojový kód určité architektury)
  - bytecode je interpretován JVM
- V praxi to s HW/OS nezávislostí není zcela pravdivé
- Problém rychlosti běhu
  - program v Javě se provádí pomaleji než obdobný program v C/C++
- Řešení
  - Těsně před provedením kompiluje kompilátor JIT (Just In Time), bytecode do strojového kódu patřičné platformy
  - A nebo počítačové platformy, kde instrukce strojového kódu = bytecode

Cíle návrhu systému

- Cíle pro uživatele
  - OS musí být snadno použitelný, snadno naučitelný, spolehlivý, bezpečný, rychlý, …
- Systémový cíl
  - OS se musí dát snadno navrhnout, implementovat, udržovat a musí být přizpůsobivý, spolehlivý, bezchybný, spolehlivý
- Zkušenost s výsledky
  - Operační systémy jsou (a asi vždy budou)
    - Obrovské – až desítky milionů řádků zdrojového kódu
    - Složité
    - Asynchronní (interaktivní)
    - (vždy) plné chyb a (často) nespolehlivé
    - Silně závislé na konkrétním hardware – obtížně přenositelné

Implementace systému

- Tradičně býval OS napsaný v symbolickém strojovém jazyku (assembleru)
- OS se stále častěji píší v běžných programovacích jazycích vysoké úrovně (obvykle C/C++)
  - lze naprogramovat rychleji
  - výsledek je kompaktnější
  - OS je srozumitelnější a lze ho snadněji ladit
  - je snadněji přenositelný na jinou architekturu

System Generation (SYSGEN)

- Operační systém je navržen tak, aby mohl běžet na jisté třídě architektur / sestav počítače
- OS musí být konfigurovatelný na konkrétní sestavu
- Program SYSGEN
  - Získává informace týkající se konkrétní konfigurace konkrétního hardwarového systému
- Bootování
  - Spuštění činnosti počítače zavedením jádra a předáním řízení na vstupní bod jádra pro spuštění činnosti
- Bootstrap program
  - Program uchovávaný v ROM, který je schopný naleznout jádro, zavést ho do paměti a spustit jeho provedení

Bootování IBM PC

- Řídí BIOS
  - provede se inicializace HW komponent
  - na základě uložené konfigurace zjistíme z kterého zařízení se má OS zavést
  - v případě pevného disku se spustí kód uložený v Master Boot Recordu (MBR)
    - tento kód například zjistí, která partition je aktivní a spustí boot sektor této partition. Kód uložený v boot sektoru načte soubory s jádrem OS do paměti
    - nebo např. LILO umožní interaktivně vybrat který OS bude zaveden (bootsektor které partition se má spustit, kde je soubor s jádrem OS)
    - tento kód může být delší než je délka MBR, musí pak být uložen v jiné oblasti disku